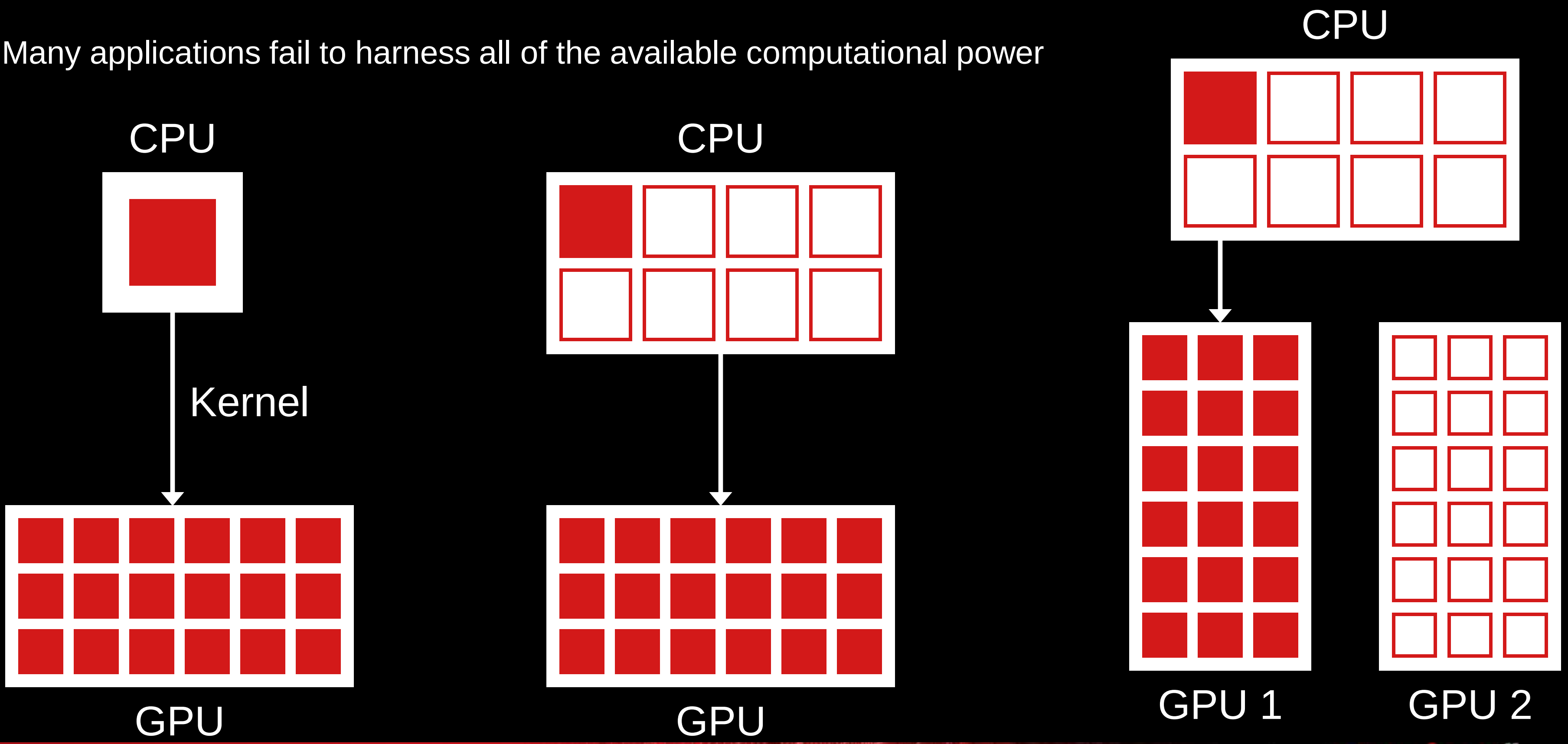
Automatic Intra-Application Load Balancing for Heterogeneous Systems

Michael Boyer, Shuai Che, and Kevin Skadron
Department of Computer Science
University of Virginia

Jayanth Gummaraju and Nuwan Jayasena
AMD

Motivation

- Many applications fail to harness all of the available computational power



Leveraging Multiple Devices

- Writing *correct* multi-device applications is challenging
- Writing *efficient* multi-device applications is even harder

- The optimal division of labor depends on:
  - Hardware characteristics
  - Input data set
  - Behavior of other applications
  - Metric being optimized

- Efficiently orchestrating the data movement and kernel invocations is complicated:
  - Set appropriate flags based on hardware and software characteristics
  - Use multiple buffers to overlap operations

Goal: Automatically load balance a single-device application across multiple (heterogeneous) devices

Outline

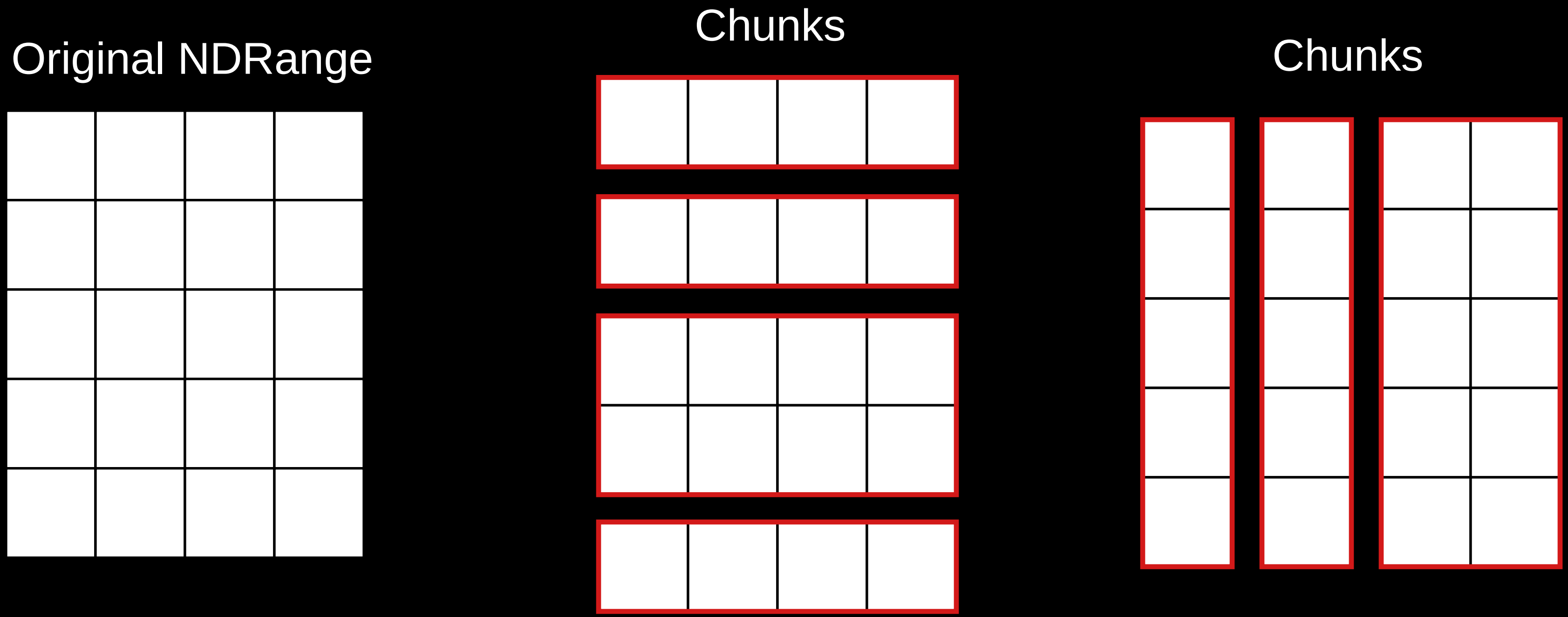
- Motivation
- Our approach: dynamic chunking
- Load balancing framework
- Preliminary results
- Conclusions and future work

Our Approach: Dynamic Chunking

- Break the kernel into multiple chunks
  - Key distinction: # chunks > # devices
- Dynamically schedule chunks to devices
  - Scheduling decisions and chunk sizes based on online profiling

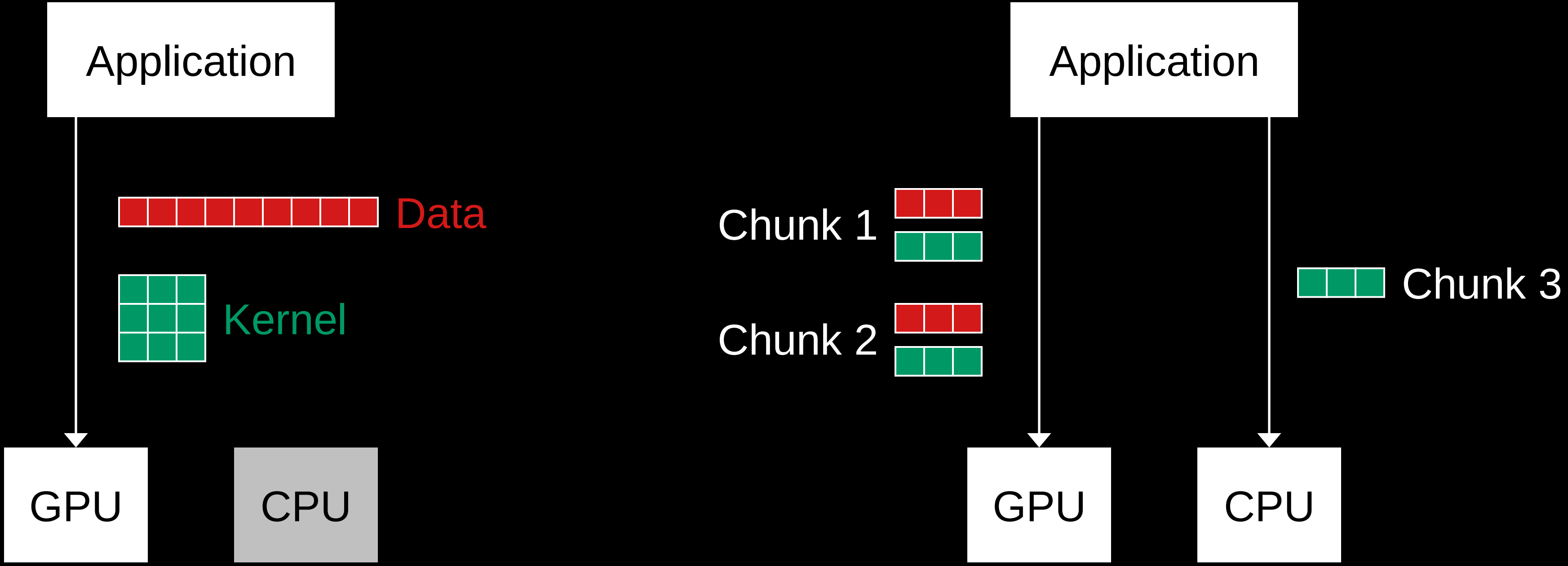
Chunking

- Chunk: a contiguous set of work groups
  - Division can be along any dimension
  - Different chunks can be different sizes



Chunking: Kernel + Data

- Need to ensure that:
  - The input data consumed by each chunk is copied to device memory
  - The output data produced by each chunk is copied back to host memory

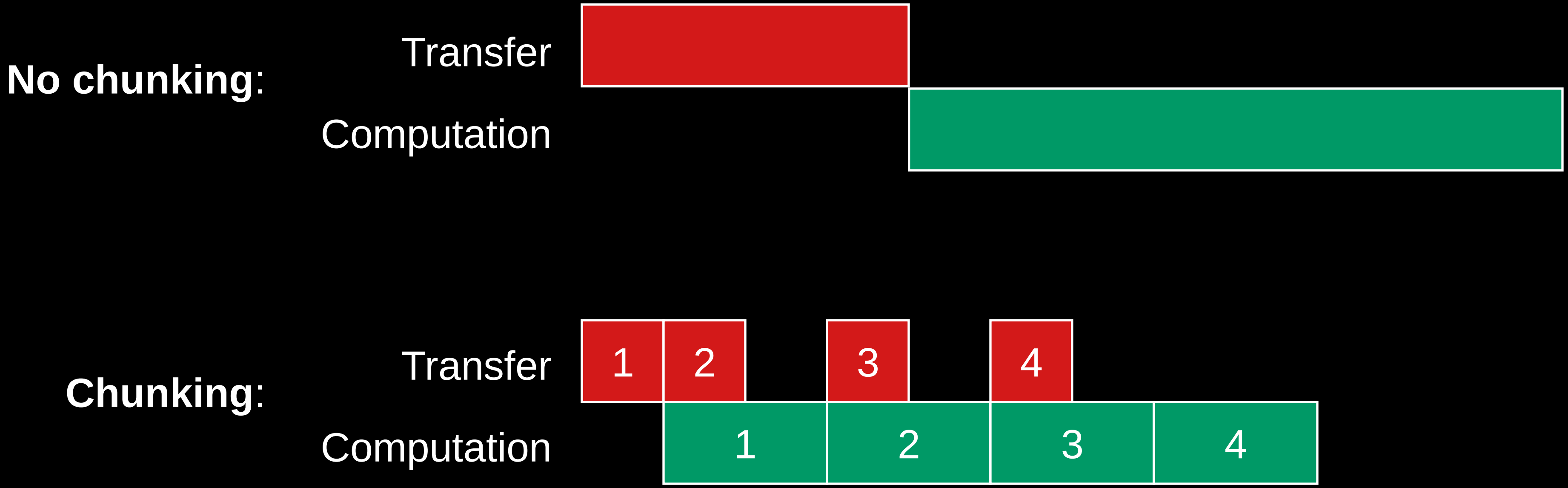


Chunking: Advantages & Challenges

- Advantages:
  - No training required, even as hardware changes
  - Can respond to dynamic performance changes due to:
    - Data-dependent behavior
    - Contention from other applications
  - Can overlap kernel execution and data transfer

- Challenges:
  - Managing contention
    - Contention from other applications
    - CPU is both host and compute device
    - Memory contention between CPU & GPU operations
  - Dispatch overhead
  - Determining which data is accessed by each chunk
    - Difficult data structures / access patterns

Chunking: Overlapping Operations

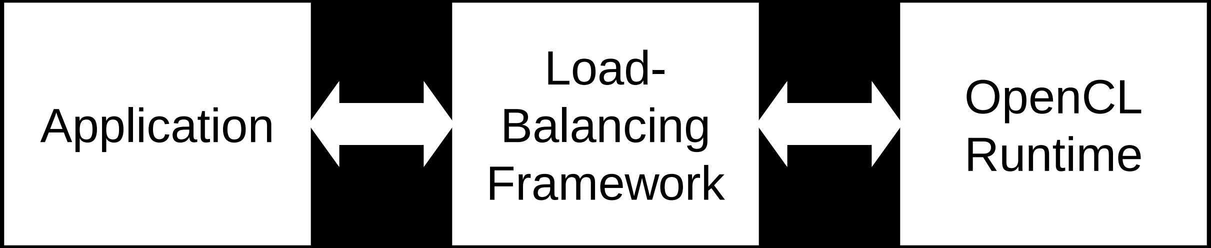


AMD Fusion11 DEVELOPER SUMMIT

Load-Balancing Framework

Framework Overview

Application View



System View

Framework Requirements

1. Intercept OpenCL API calls
2. Determine what data to transfer to each device
3. Orchestrate and balance execution across the devices

Framework Components

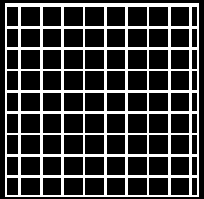
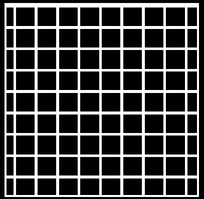
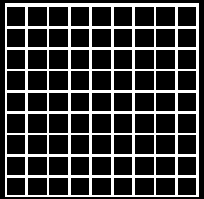
1. API intercept layer
   – Intercepts and transforms calls from application to OpenCL runtime

2. Access pattern extractor
   – Analyzes kernel source code to extract data access patterns

3. Chunk scheduler
   – Breaks kernels into chunks and schedules them onto devices

API Intercept Layer

- Intercepts each OpenCL API call and replicates it across multiple devices or transforms it

OpenCL API Function	Application View	System View	
clCreateCommandQueue	GPU 	GPU 	CPU 
clCreateBuffer	GPU 	GPU 	CPU 

Access Pattern Extractor

- Need a mechanism for sending the right data to each device

- Given the kernel source, determines:
  - Mapping from chunk to memory region
  - Preferred chunking direction
- Kernel source code is available from intercepting call to the OpenCL compiler

- Output: function that returns the region of memory accessed by a given chunk
  - Callable by the chunk scheduler
  - Works for arbitrary chunk sizes

Access Pattern Extractor: Details

- Built on Clang (LLVM’s front-end)
  – Clang’s OpenCL support is under active development
  – For now, add an implicit header to define built-in data types and functions

- Basic idea: traverse the kernel AST
  – Identify accesses to memory buffers
  – Express buffer offsets in terms of values that can be reasoned about at kernel invocation time
  – Determine relationship between accesses from different work items

Access Pattern Extractor: Example

```
__kernel void blackScholes(const __global float4 *randArray,
                                        …
                                        __global float4 *put, int width) {

  size_t xPos = get_global_id(0);
  size_t yPos = get_global_id(1);
  float4 inRand = randArray[ yPos * width + xPos ];
  …
  put[ yPos * width + xPos ] = KexpMinusRT * phiD2 - S * phiD1;
```

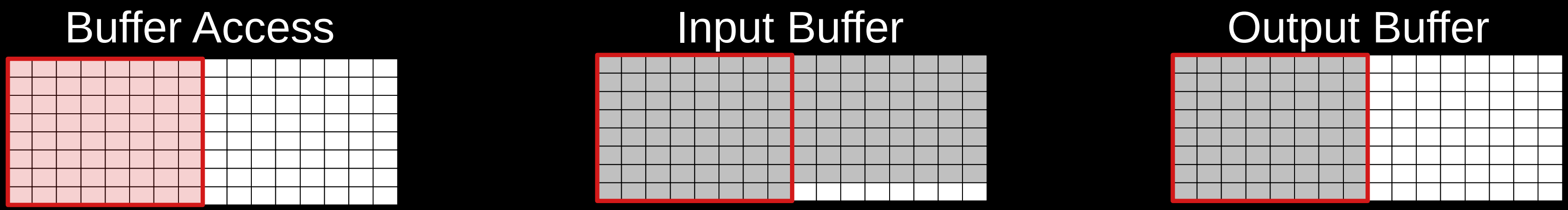
Read access

Write access

Access Pattern Extractor: Input vs. Output Buffers

- Input buffers:
  – Can afford to be imprecise
  – Approach: determine minimum and maximum offset and transfer entire range

- Output buffers:
  – Need to be precise
  – Approach: determine parameters of strided access



Scheduler

- Breaks kernel into chunks
- For each chunk:
  – Sends input data to device
  – Launches kernel on device
  – Copies output data back to host

- Mapping of chunks to devices is determined dynamically, based on online profiling data

Dynamic Scheduling

- If number of work groups is small, skip chunking and send whole kernel to one device

- Otherwise, send initial chunks to each device:
  - Initial chunk size set to exactly fill each device
- Maintain two chunks outstanding to each device to hide dispatch overheads
- Exponentially increase chunk sizes for “fast” devices until “slow” device has completed a chunk
- Once performance data is available for all devices:
  - Distribute a portion of the remaining work to all devices based on relative performance
  - Maintain aggregate history information, but decay it exponentially

Preliminary Results

Experimental Systems

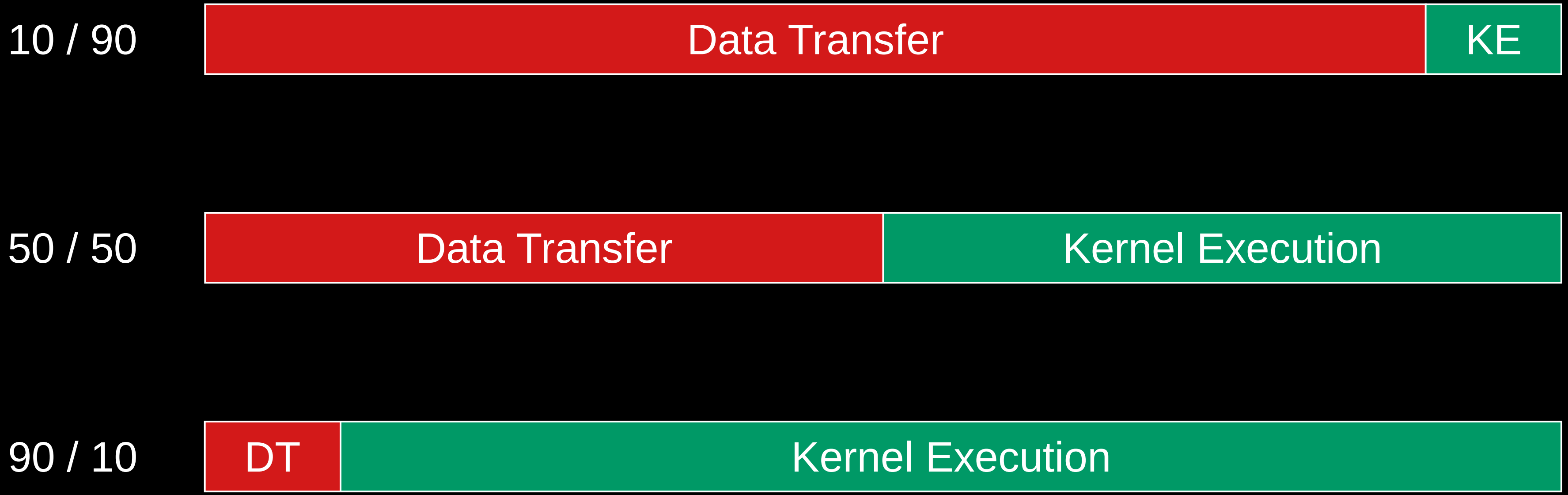
1. CPU + GPU:
   - AMD Radeon HD 5870 (Cypress)
   - Intel Core i7 920: quad-core, hyper-threaded, 2.67 GHz

2. Homogenous Multi-GPU:
   - 2 x AMD Radeon HD 5870

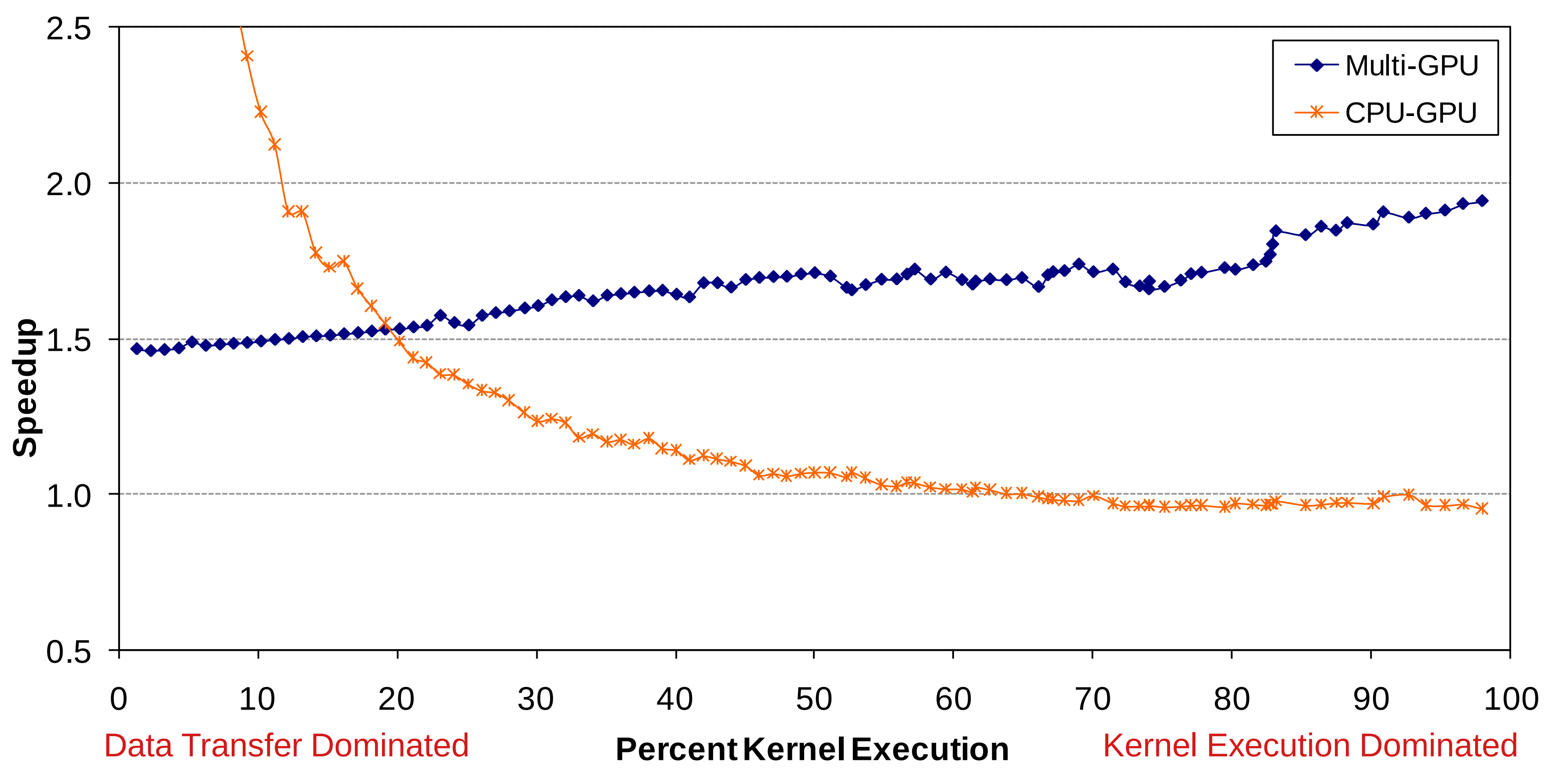
3. Heterogeneous Multi-GPU:
   - AMD Radeon HD 5870
   - AMD Radeon HD 6570 (Turks)

Synthetic Benchmark: Computation-to-Communication Ratio

- Ratio of kernel execution time to data transfer time can be controlled arbitrarily



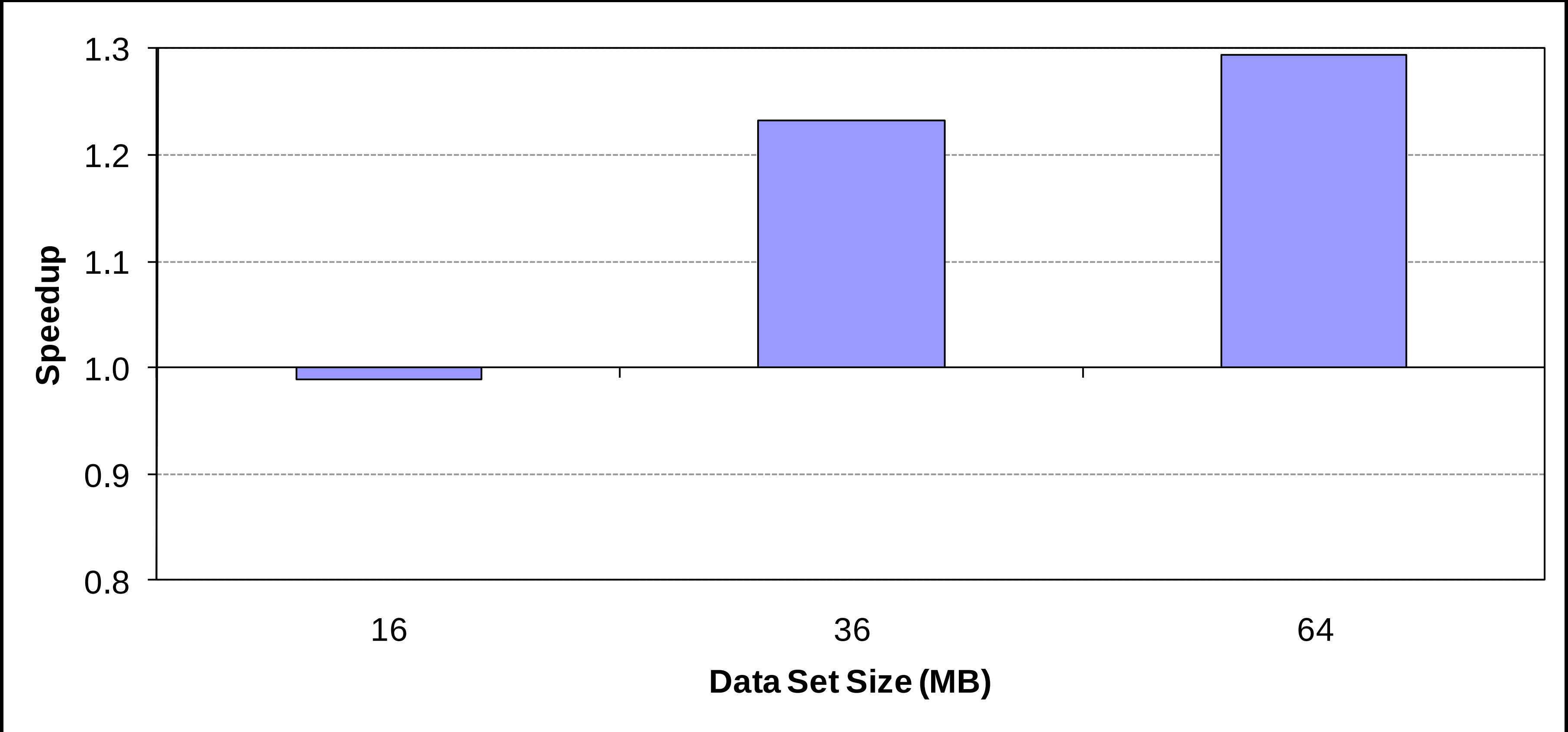
Synthetic Benchmark Results



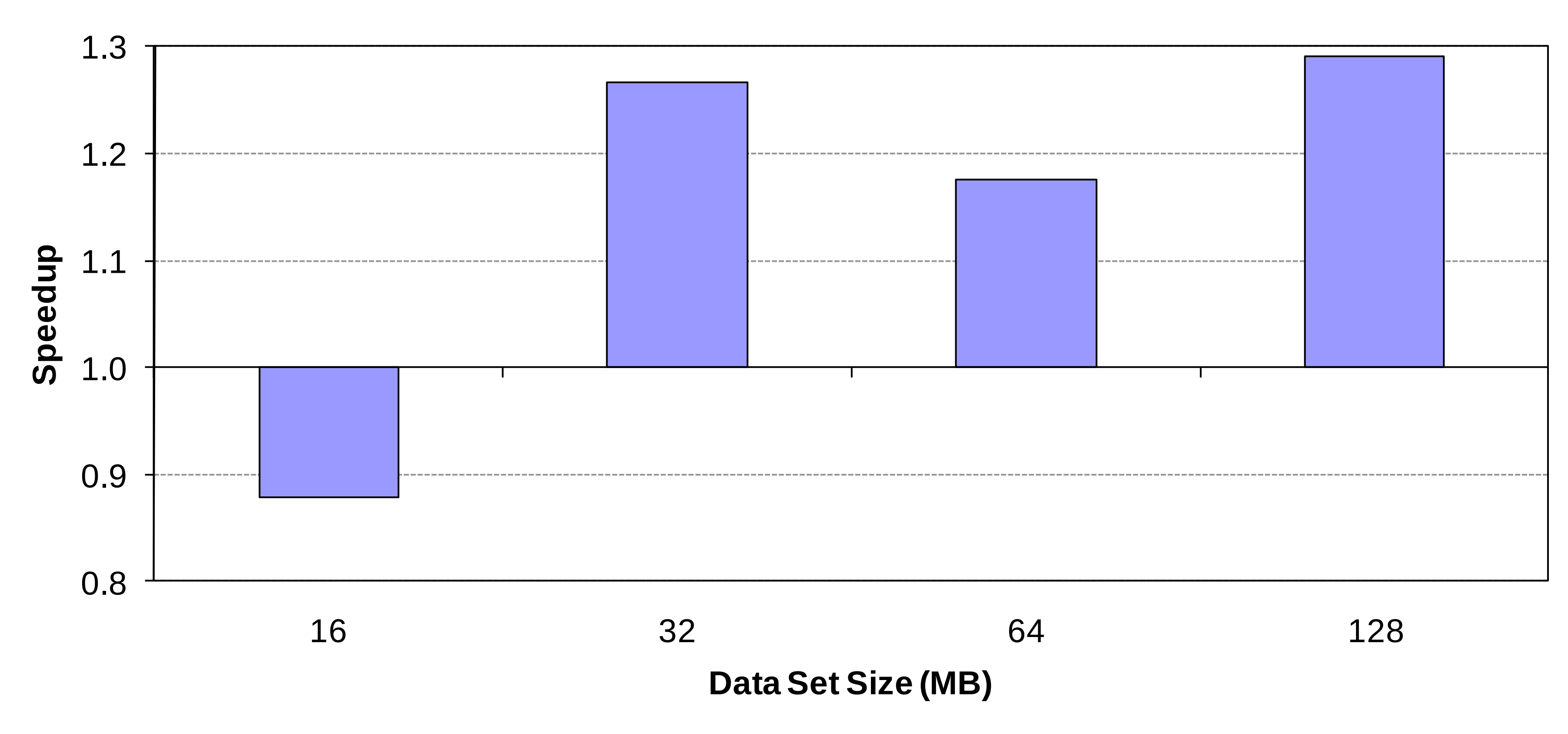
Sample Applications

- From Rodinia benchmark suite:
  - SRAD
- From AMD APP SDK:
  - Mandelbrot
  - Black-Scholes

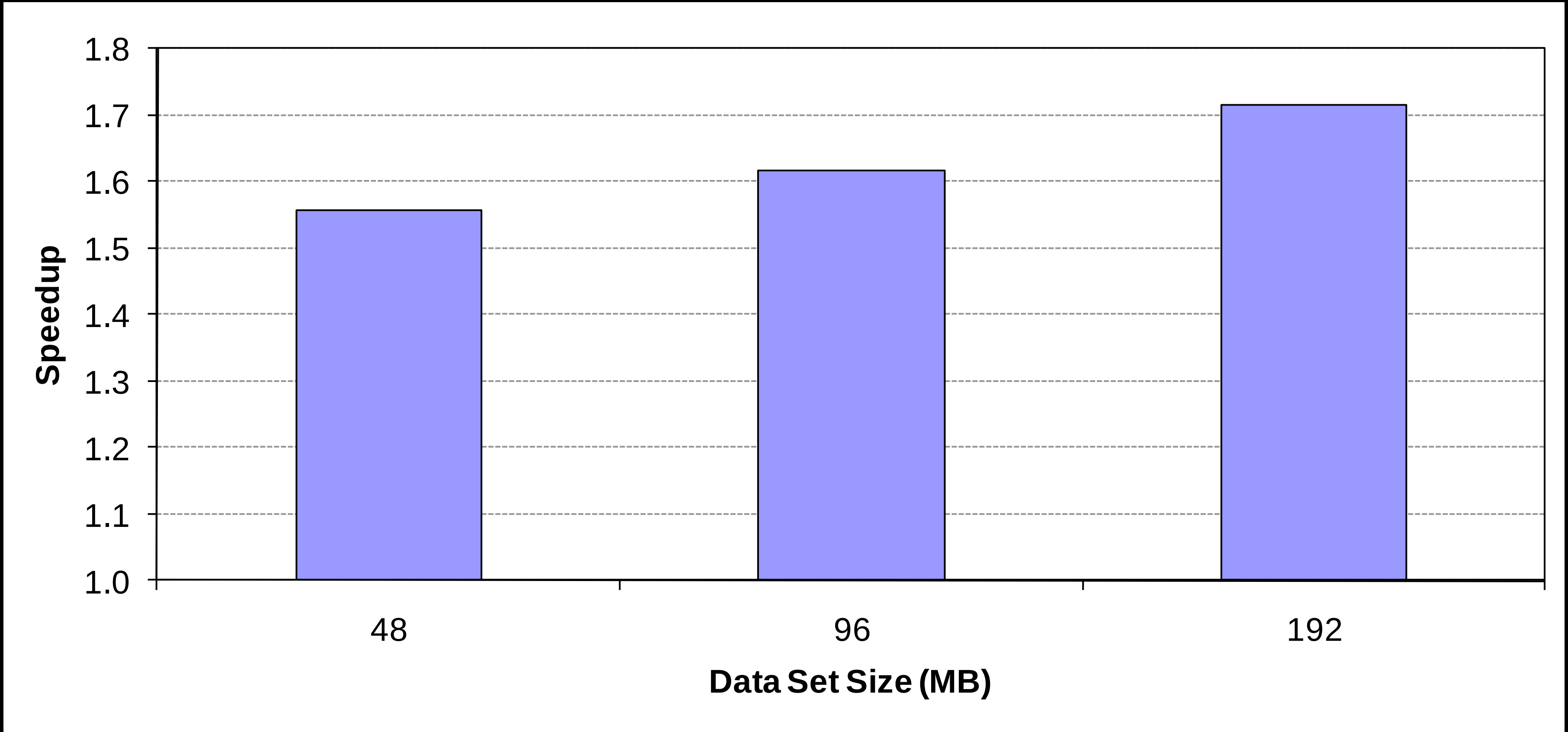
Results: CPU + GPU, SRAD



Results: Heterogeneous Multi-GPU, Mandelbrot



Results: Homogeneous Multi-GPU, Black-Scholes*



Previous Work: Qilin

- Divides a CUDA kernel across a CPU and GPU

- Limitations:
  - Requires manual creation of CPU & GPU versions of kernel
  - Requires a training phase
  - Scheduling is static
  - Only works on NVIDIA GPUs

- Reference: C.-K. Luk, S. Hong, and H. Kim, “Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping,” in *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2009.

Previous Work: Single Compute Device Image

- Divides an OpenCL kernel across multiple GPUs

- Limitations:
  - Naïve scheduling: each device gets an equal amount of work
  - Only works on NVIDIA GPUs

- Reference: J. Kim, H. Kim, J. H. Lee, and J. Lee, “Achieving a single compute device image in OpenCL for multiple GPUs,” in *Proceedings of the 16th ACM symposium on Principles and Practice of Parallel Programming*, 2011.

Challenges and Future Work

- Tune the framework for different hardware and software configurations
- Optimize for different metrics
- One version of the kernel for multiple devices
  - Optimizations for GPU may hurt performance on CPU and vice versa
  - Supporting device-specific kernels would allow a tradeoff between programmer effort and performance
- Multiple kernel calls
  - Need to understand data flow between kernel calls
- Possible (but rare) for work groups to communicate with each other using atomic instructions
  - Difficult to support across multiple devices efficiently
- Deployment possibilities:
  - OpenCL layer targeting standalone applications
  - Layer in the Fusion software stack

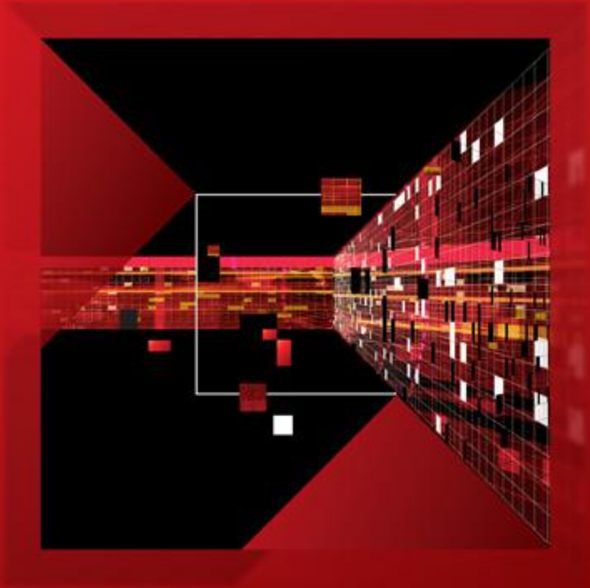
Conclusions

- Heterogeneous multi-device systems (like Fusion) are becoming ubiquitous
  - Effectively utilizing the available devices is difficult

- Our framework automatically load balances unmodified OpenCL applications across multiple (possibly heterogeneous) devices
  - Extracts access patterns to determine mapping of work groups to data
  - Uses online profiling to guide scheduling decisions

- Preliminary performance results are encouraging

Questions



Disclaimer & Attribution

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. There is no obligation to update or otherwise correct or revise this information. However, we reserve the right to revise this information and to make changes from time to time to the content hereof without obligation to notify any person of such revisions or changes.

NO REPRESENTATIONS OR WARRANTIES ARE MADE WITH RESPECT TO THE CONTENTS HEREOF AND NO RESPONSIBILITY IS ASSUMED FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

ALL IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE ARE EXPRESSLY DISCLAIMED. IN NO EVENT WILL ANY LIABILITY TO ANY PERSON BE INCURRED FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

AMD, the AMD arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. All other names used in this presentation are for informational purposes only and may be trademarks of their respective owners.

The contents of this presentation were provided by individual(s) and/or company listed on the title page. The information and opinions presented in this presentation may not represent AMD's positions, strategies or opinions. Unless explicitly stated, AMD is not responsible for the content herein and no endorsements are implied.